Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Robot móvil con ROS para la lectura de códigos de barras mediante zoom óptico en un entorno comercial

Autor: Pedro Tito Macías Roselló

Tutor: Miguel Ángel Ridao Carlini

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Robot móvil con ROS para la lectura de códigos de barras mediante zoom óptico en un entorno comercial

Autor:
Pedro Tito Macías Roselló

Tutor:
Miguel Ángel Ridao Carlini
Profesor Titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Robot móvil con ROS para la lectura de códigos de barras mediante zoom óptico en un entorno comercial

Autor: Pedro Tito Macías Roselló
Tutor: Miguel Ángel Ridao Carlini

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi familia
A mis amigos
A mi clan

Agradecimientos

Me gustaría comenzar estas lineas agradeciendo a mi familia el apoyo incondicional a mis estudios los últimos años, especialmente cuando aportaban la motivación que yo no era capaz de encontrar.

También quiero mostrar mi agradecimiento a todos mis compañeros en la Escuela, que me han enseñado el verdadero significado del trabajo en equipo, a mi clan por ayudarme a superar las dificultades con alegría, a mis amigos por recordarme el valor de dedicarme tiempo a mi mismo y a Esther por inspirarme siempre a ser la mejor versión de mí mismo.

Por último, quiero agradecer también la labor docente de los profesores que han logrado transmitirme su ilusión por el conocimiento, a mi tutor Miguel Ángel por brindarme la oportunidad de trabajar en un proyecto tan completo y a Manuel Mora por haber sido un guía para mi en este proyecto.

Pedro Tito Macías Roselló

Sevilla, 2021

Resumen

Este proyecto surge a partir de la propuesta de la empresa Tier1 de continuar con el desarrollo de una plataforma robótica móvil para entornos de venta al por menor. Antes de comenzar este proyecto el robot era capaz de realizar un mapa de una espacio cerrado de forma autónoma, localizar productos representados por códigos QR y desplazarse hacia ellos posteriormente.

El objetivo de esta intervención es doble. Por un lado, se pretende mejorar la calidad del sistema de visión del robot para permitirle escanear códigos de barras, en concreto del tipo EAN-13, que requieren de una resolución de imagen mayor que la de los códigos QR, así como establecer una conexión con un servidor externo de Tier1 a través de una API para almacenar en la memoria del robot el nombre de los productos correspondientes a los códigos escaneados. Para ello, se incorpora al sistema una cámara PTZ, con movimimiento en dos ejes y zoom óptico.

Simultáneamente, se quiere mejorar la documentación y organización de todo el proyecto que, en el momento de comenzar este trabajo, contaba con cientos de archivos de códigos fuente y muchos de ellos obsoletos y sin ninguna explicación.

Abstract

This project is born from the proposal of the company Tier1 to continue with the development of a robotic mobile platform for the retail sector. Before the beginning of this project the robot was able to autonomously create a map of a closed space, locate products represented with QR codes and move towards them afterworks.

The objetive of this intervention is dual: On one side, the intention is to improve the quality of the vision system of the robot to allow it to scan barcodes, specifically of the EAN-13 type, which require am image resolution greater than the one needed for QR codes, as well as stablish a connection with an external server from Tier1 through an API to store in the robot's memory the name of the products related to the scanned codes. For this, a PTZ camera with movement in two axis and optical zoom is added to the system.

Simultaneously, we want to improve the documentation and organization of the entire project that, at the beginning of this work, was made of hundreds of source ode files, a lot of them obsoletes and lacking of any explanation.

Índice

1 Introducción

La idea para desarrollar el proyecto que aquí se expone surge como una propuesta por parte de la empresa Tier1 en colaboración con la Escuela Técnica Superior de Ingeniería de Sevilla en 2016. En ese momento, se planteó el desarrollo de una plataforma robótica móvil orientada al sector de la venta al por menor que pudiera desplazarse por un local e ir llevando a la caja los productos de una lista dada.

El proyecto se desarrolló durante un tiempo y varios estudiantes trabajaron en él implementando nuevas funcionalidades, pero desde 2017 su desarrollo se detuvo durante unos años y ahora, en 2021, se ha retomado el trabajo, por lo que antes siquiera de poder proponer soluciones a los nuevos problemas planteados, fue necesario invertir tiempo y esfuerzo en buscar, estructurar y comprender todo el código fuente sobre el que se sostenía la plataforma, así como los dispositivos electrónicos que lo componían y las tecnologías que utilizaba.

Con el fin de comprender el contexto en el que se ha desarrollado este sistema se hace imprescindible mencionar los trabajos de Francisco Javier Buenavida Durán [6] y Adrián Fernández Sola [9], dos de los estudiantes que trabajaron previamente en este proyecto y cuyos TFG han resultado de gran utilidad para entender como se había estructurado previamente el funcionamiento del robot.

2 Marco teórico

En el momento en el que se empezó a desarrollar este trabajo, el sistema ya contaba con una serie de funcionalidades que llevaba a cabo sin problemas, las cuales se describen a continuación.

2.1 Alcance

En los trabajos de Adrián Fernández Sola[9] y Francisco Javier Buenavida Durán [6] se explica con detalle el desarrollo de todo el trabajo previo en este proyecto, sin embargo, a continuación se exponen algunos puntos claves de las capacidades de la plataforma turtlebot antes de la intervención.

2.1.1 Generación autónoma de mapas

Gracias a la información obtenida por el sensor lidar y el uso de un algoritmo de exploración, el robot puede desplazarse por un espacio cerrado y generar un fichero que con información sobre los obstáculos de dicho espacio.

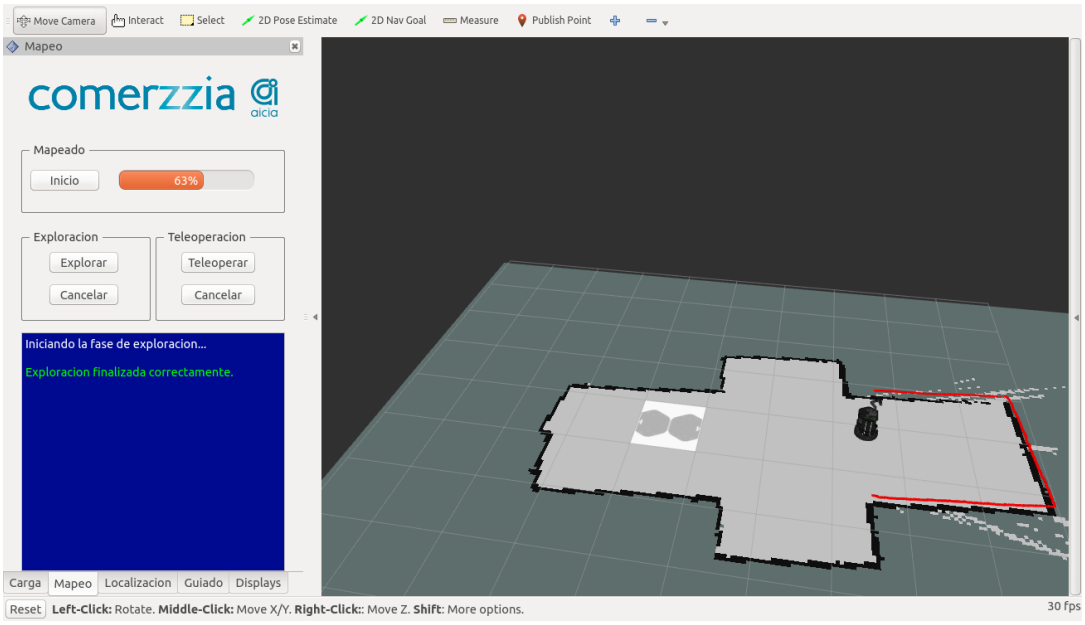


Figura 2.1 Fase de obtención del mapa.

2.1.2 Localización de productos

Una vez obtenido el mapa, el robot puede localizar productos representados por un código QR, bien de forma autónoma utilizando el algoritmo desarrollado para este fin [9], o bien recibiendo órdenes manualmente

para escanear una zona del mapa en concreto. Después, almacena las coordenadas tridimensionales de estos productos en un fichero.

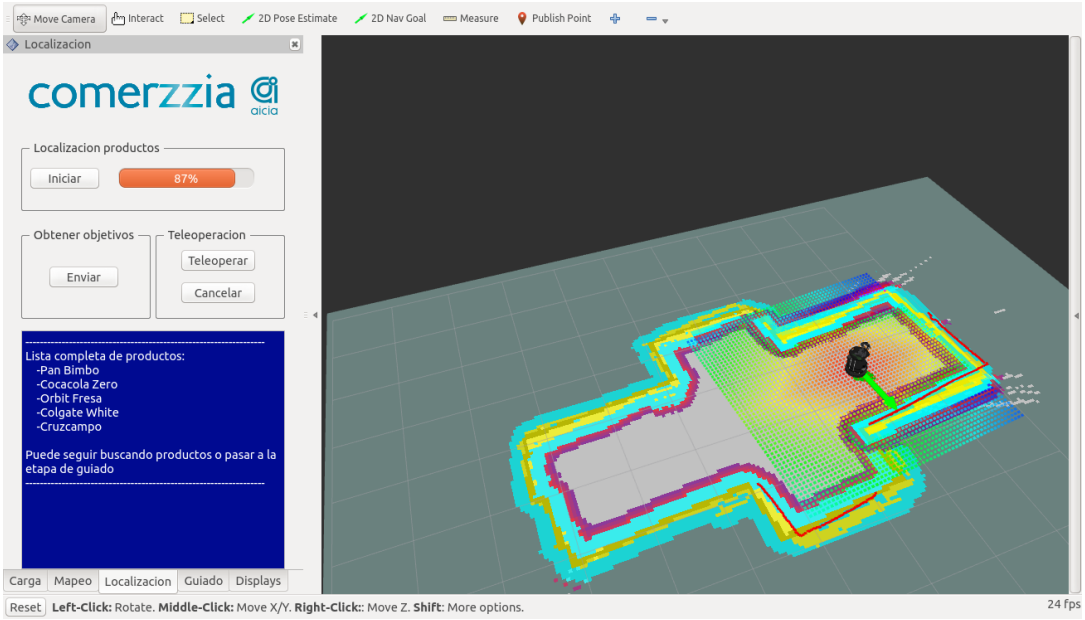


Figura 2.2 Fase de localización de productos.

2.1.3 Guiado

Por último, es posible enviarle al robot un listado de productos a buscar, el cuál compara con el fichero de productos creado previamente y, si existe, se desplaza a ellos uno a uno.

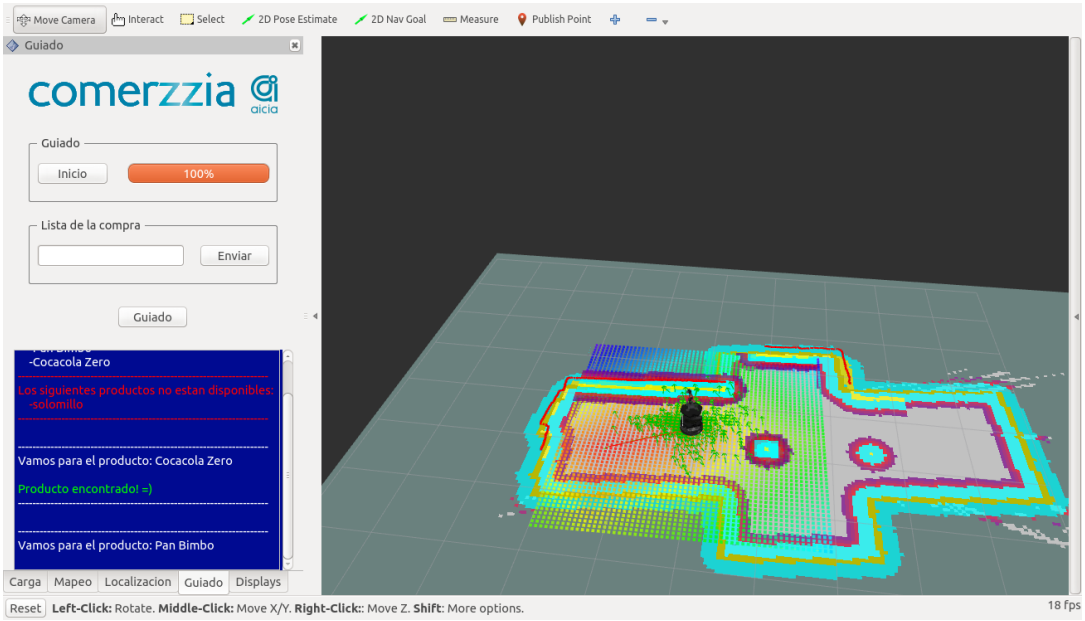


Figura 2.3 Fase de guiado hacia los productos.

Originalmente para esta fase se implementó un pequeño brazo robótico en la plataforma [6], pero fue desechado por su limitado alcance y falta de potencia para cargar con la mayoría de productos.

2.2 Limitaciones

Para evaluar las dificultades a las que se enfrenta este sistema, es necesario tener en cuenta cual es la finalidad para la que fue concebido, es decir, el sector de las ventas al por menor (en inglés "retail"), donde el consumidor se desplaza libremente por un espacio en el que los productos están dispuestos en estanterías sobre sus correspondientes códigos de barras.

Este entorno plantea dos problemas para el sistema existente. Por un lado, el algoritmo de localización de productos ha sido desarrollado para leer códigos QR, no códigos de barras, los cuales normalmente requieren de una precisión mayor.

Por otro lado, en un comercio real, el código correspondiente a cada producto no codifica el nombre del mismo, sino un número asociado al mismo en la base de datos del vendedor, por lo que el robot, a priori, no podría proporcionar al usuario los nombres de los productos que ha localizado.

2.2.1 Códigos de barras y QR

Para comprender por qué es necesario plantear una solución específica para el reconocimiento de los códigos de barras, es necesario entender las diferencias y similitudes entre códigos de barras lineales y códigos QR.

Por un lado, los códigos de barras lineales (Figura 2.4), patentados en 1952 y popularizados en la década de los 80, fueron ideados en una época en el que la posibilidad de utilizar cámaras para la decodificación de los mismos era impensable, por lo que se diseñaron como un sistema capaz de codificar combinaciones cortas de caracteres que pudieran ser interpretadas con un lector láser que realiza un barrido en una sola dimensión.

Figura 2.4 Código de barras en formato EAN-13.

En sus comienzos, resultaron una solución extremadamente útil para acelerar las tareas de inventario, siendo su mayor limitación la necesidad de acercar el sensor a menos de 15cm en la mayoría de los casos.

Por otro lado, los códigos QR (Figura 2.5) surgieron en 1994 como una evolución natural del código de barras, con la idea de aprovechar la creciente capacidad de las cámaras digitales, permitiendo así codificar la información en patrones bidimensionales, eliminando restricciones como tener que leer el código de forma totalmente perpendicular y aumentando la distancia a la que los sensores podía interpretarlos, siendo la única limitación la resolución y nitidez del sensor.

Figura 2.5 Código QR.

Esta última ventaja radica, principalmente, en que la información en un código QR no se codifica según el tamaño de las figuras que lo componen, como sí lo hacen los códigos de barras lineales, sino según la posición de éstas. Por lo tanto, dentro de un mismo código, si existe resolución suficiente para percibir un solo módulo, se podrá interpretar toda la información. En los códigos de barras lineales, sin embargo, el tamaño de los módulos puede ser de hasta 8 veces mayor de uno respecto a otro (Figura 2.4), lo que puede hacer algunas partes del mismo imposibles de interpretar a ciertas distancias, pudiendo resultar en decodificaciones erróneas.

Estas nuevas capacidades permitieron que el código QR se convirtiera en una herramienta extremadamente útil como marcador para los algoritmos de visión artificial que utilizan muchos robots actualmente (**??**).

Figura 2.6 Cámara del turtlebot interpretando un código QR.

En conclusión, los códigos de barras lineales y QR, a pesar de poder codificar información de forma similar, necesitan de sensores y condiciones muy diferentes para poder ser interpretados con precisión. No obstante, cuando la nitidez de la imagen tomada por una cámara es lo suficientemente alta, ambos códigos pueden ser interpretados sin problema (Figura 2.7).

Figura 2.7 Cámara del turtlebot interpretando un código QR y un código de barras simultáneamente.

2.2.2 Codificación de la información

Para esta aplicación concreta, es necesario utilizar códigos de barras en el formato EAN-13, que es un estándar internacional de códigos de barras que codifica un número de 13 dígitos y es el que utiliza la empresa que encarga el proyecto en sus establecimientos.

Codificar solo 13 números significa que no se puede almacenar el nombre completo de un producto en el código de barras, que es la información que la aplicación muestra al usuario final. Por lo tanto, el robot debería de tener almacenada toda esa información en su memoria interna. Esto supone que, para cada entorno en el que funcione este sistema, sería necesario cargarle manualmente en la memoria la base de datos, lo cual aumentaría la complejidad de su operación para el usuario final.

La otra alternativa es que el robot acceda a la base de datos del establecimiento remotamente, ya que todos los comercios convencionales que utilizan códigos de barras ya cuentan con esta información, y la única configuración previa por parte del usuario sería la de configurar una conexión a internet.

2.3 Nuevos Objetivos

A raíz de las limitaciones que se han observado sobre el sistema, la empresa responsable plantea principalmente dos objetivos a cumplir, que facilitarán la implementación del robot en entornos reales:

- Incluir la capacidad de leer códigos de barras con la misma facilidad que códigos QR en la fase de localización de productos.
- Establecer una comunicación entre el robot y la base de datos del comercio para obtener los nombres de los productos a partir de su código numérico.

Además, al tener en cuenta la falta de uniformidad y documentación en el proyecto y previendo futuras mejoras sobre el mismo, se plantea un tercer objetivo que se llevará a cabo de forma transversal durante todo el proceso de desarrollo:

- Depurar y trasladar el proyecto a un sistema de control de versiones.

En este último caso, aunque se trate de una tarea que ha tomado mucho tiempo llevar a cabo, resulta poco relevante enumerar todos los pasos realizados para la limpieza de los ficheros del proyecto, ya que ha consistido principalmente en eliminar ficheros obsoletos, añadir líneas de comentarios y crear un repositorio online. En cualquier caso, como prueba de este trabajo, se adjunta el Apéndice A, que contiene un manual actualizado del proceso de instalación del proyecto.

3 Hardware

A continuación se detallan los componentes electrónicos que conforman el robot y una descripción de sus características más relevantes para este trabajo. Cabe destacar la cámara PTZ como la pieza clave en esta intervención ya que, como se explica más adelante, sobre ella se sustentan las nuevas funcionalidades implementadas en la plataforma y es el único componente que no estaba presente en versiones anteriores.

3.1 Plataforma turtlebot 2

La plataforma TurtleBot [4] es un robot móvil básico, de código abierto e ideado con la finalidad de facilitar el desarrollo en el campo de la robótica, concretamente utilizando herramientas de software como ROS (en la que se profundizará más adelante). Al tratarse de un proyecto cuyo software y hardware son libre, cuenta con una activa comunidad en internet que desarrolla múltiples paquetes de aplicaciones de libre acceso, lo que lo convierte en el entorno idea para desarrollar sistemas experimentales como este.

Figura 3.1 Plataforma TurtleBot 2.

Para este sistema en concreto se cuenta con la segunda versión de esta plataforma, TurtleBot 2 (Figura 3.1).

3.1.1 Base Kobuki

De entre los componentes de esta plataforma robótica básica, el principal es su base (Figura 3.2), desarrollada por Kobuki, que cuenta con cuatro ruedas, dos de ellas motrices dirigidas por dos motores independientes y las otras dos pasivas, además de una batería, un sensor de inercia y un microcontrolador para realizar tareas de control de velocidad, odometría y monitorización del estado de la batería. Esta configuración aporta al robot una gran estabilidad y maniobrabilidad en espacios pequeños.

Figura 3.2 Base de la plataforma TurtleBot 2.

Tabla 3.1 Características de la plataforma TurtleBot 2 .

Velocidad máxima	2.34 Km/h
Capacidad de carga	5 Kg
Capacidad de la batería	4400 mAh
Autonomía	5 h

3.1.2 Cámara de profundidad Kinect V2

Originalmente este proyecto utilizaba la primera versión de la cámara Kinect de Microsoft, un dispositivo que combinaba un sensor de imagen con un sensor de profundidad (Figura 3.3) y que fue lanzado al mercado en 2010. Al ser un producto producido en serie para el sector de los videojuegos, su precio era muy inferior a cualquier otra alternativas existentes en aquel momento. Además, contaba con una fuerte comunidad en internet que desarrollaba controladores para utilizarla en aplicaciones de robótica.

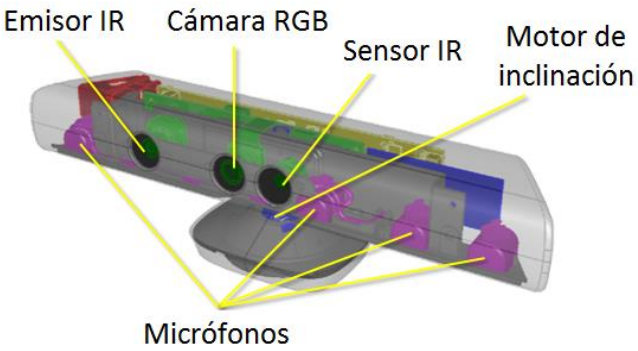


Figura 3.3 Primera versión de la cámara Kinect.

Todas estas ventajas se mantuvieron en la segunda versión del dispositivo, Kinect V2, que fue lanzada en 2013 y mejoró las características de la versión anterior (Tabla 3.2)

Al ofrecer una resolución tanto de imagen como de profundidad mucho mayor, la Kinect V2 sustituyó a la versión anterior y es con la que se desarrollaron las últimas fases del proyecto.

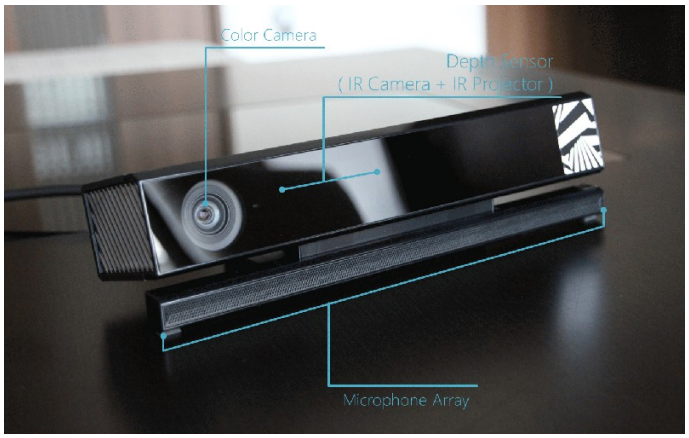


Figura 3.4 Segunda versión de la cámara Kinect.

Tabla 3.2 Características de la cámara Kinect V2 [8].

Resolución del sensor infrarojo	512x424 píxeles
Resolución de la cámara RGB	1920x1080 píxeles
Ángulo de visión	70ºx60º
Fotogramas por segundo	30 fps
Rango de distancia operativa	0.5 - 4.5 m
Object Pixel Size	entre 1.4 mm (a 0.5 m) y 12 mm (a 4.5 m)

3.1.3 Lidar Hokuyo

Otra parte clave de la plataforma es su sensor Hokuyo URG-04LX-UG01 (Figura 3.5), un sensor de barrido láser que permite que el robot identifique con suficiente precisión los obstáculos que no pueda atravesar.

Figura 3.5 Sensor de barrido láser.

Tabla 3.3 Características del sensor Hokuyo URG-04LX-UG01 [9].

Peso	160 g
Consumo	2.5 W
Distancia máxima	4 m
Ángulo de barrido	210 º
Error	3 %
Resolución angular	0.352 º

3.2 Intel NUC

Como unidad central para realizar todos los cálculos y tratar todos los datos con los que trabaja el robot, se utiliza un ordenador Intel NUC NUC5i5RYH (Figura 3.6), que es un modelo especialmente compacto, ligero, con un consumo reducido y una potencia más que suficiente para desarrollar las tareas de computación del robot.

Figura 3.6 PC Intel NUC.

Tabla 3.4 Características ordenador Intel NUC[9].

Procesador	Intel Core i5-5250U
Potencia	15 W
Memoria RAM	4 Gb
Memoria de alamcenamiento	120 Gb

3.3 Cámara PTZ Axis M5054

A partir de este punto, los componentes de hardware descritos forman parte de la propuesta planteada en este trabajo para cumplir con los objetivos previamente. Por lo tanto, se profundizará más en los detalles técnicos y en la justificación para implementar estos componentes en concreto.

La cámara Axis M5052 que se utiliza en este proyecto es un modelo comercial de cámara PTZ (Pan-Tilt-Zoom), llamadas así por disponer de movimiento de rotación en dos ejes (pan & tilt) y un zoom variable que puede ser digital u óptico. Este tipo de cámaras son muy comunes a la hora de instalar equipos de videovigilancia, ya que permiten a un operario modificar el encuadre de la imagen según sea necesario sin acceder físicamente al equipo, aumentando considerablemente la cantidad de detalles que se pueden observar desde un solo dispositivo.

Sin embargo, para comprender en profundidad por qué este dispositivo supone una mejora considerable en cuanto a las prestaciones del sistema respecto a la cámara Kinect V2, se hace necesario aclarar la diferencia entre zoom digital y zoom óptico.

3.3.1 Zoom óptico vs zoom digital

Partiendo desde la base de que, en el contexto de las cámaras convencionales, el término zoom se utiliza para hacer referencia a un aumento del tamaño con el que se perciben los objetos de la imagen, podemos tratar a

la versión digital y la óptica como dos formas de afrontar el mismo problema, cada una con sus ventajas y desventajas.

Por un lado, el zoom óptico basa su funcionamiento en el comportamiento físico de la luz desde el punto de vista de la óptica. Desde el siglo XIX, las cámaras fotográficas han aprovechado las propiedades de las lentes esféricas para concentrar la luz en una superficie fotosensible y, de este modo, transferir la imagen del mundo real bien a un negativo analógico o a un sensor digital (Figura 3.7). A día de hoy, aunque en un contexto más sofisticado, el principio de funcionamiento de las cámaras, tanto en fotografía como en vídeo, es el mismo.

De este modo, modificando la distancia entre la lente y el sensor (lo que se conoce como *distancia focal*) se puede lograr alterar el tamaño aparente de los objetos en la imagen final.

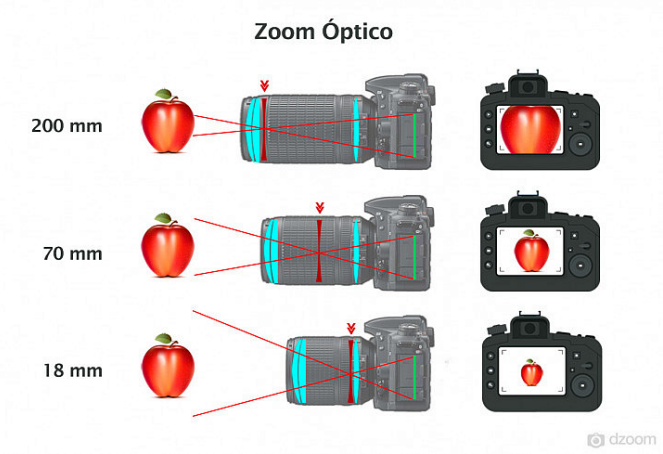


Figura 3.7 Formación de la imagen en un sensor digital [5].

Por otro lado, el zoom digital es una técnica mucho más moderna y forma parte del proceso de edición de una imagen, es decir, no se aplica en el momento de la toma de información, sino que modifica la información obtenida previamente. También modifica el tamaño aparente de los objetos en una imagen, pero lo hace aumentando individualmente el tamaño de todos los píxeles que la componen, por lo tanto, en ningún momento se obtendría más información de la que ya existe.

En comparación, el zoom óptico puede llegar a aportar una mayor información de un objeto que se está alejado, pero para utilizar el necesaria una cámara que cuente con una distancia focal ajustable, mientras que el zoom digital es aplicable en cualquier caso, pero no permite obtener más información (Figura 5.2). Es por esta razón que se ha estimado necesario incluir una cámara con zoom óptico al proyecto, ya que permitirá obtener información de códigos de barras mucho más pequeños que anteriormente.

(a) Zoom óptico. **(b)** Zoom digital.

Figura 3.8 Comparación entre zoom digital y zoom óptico.

3.3.2 Requisitos especiales

A pesar de las ventajas a nivel de nitidez que ofrece ese dispositivo, su implementación en el sistema no es trivial, ya que su diseño no está orientado al campo de la robótica, donde su posición es variable y puede estar

alimentado por una batería, sino a la videovigilancia, donde mantiene una posición física y recibe potencia eléctrica directamente desde la red.

Estándar 802.3af (PoE)

La mayor dificultad que plantea la instalación de la cámara Axis M5054 es el estándar PoE [7], que si bien se incluyó en su diseño como forma de reducir el número de cables en su instalación, para esta aplicación es necesario plantearlo de una forma distinta.

El protocolo PoE (Power Over Ethernet) recogido en el estándar 802.3af de la IEEE [7] recoge las directrices para implementar en los dispositivos electrónicos que utilizan protocolos de red la posibilidad de recibir potencia eléctrica a través del mismo puerto por el que intercambian información, el RJ-45. Así, los dispositivos que utilizan esta tecnología, pueden utilizar un solo cable hacia un switch ethernet compatible para comunicarse con la red y, al mismo tiempo, recibir de la misma la potencia necesaria para funcionar.

Inyector PoE

La plataforma robótica que se utiliza en este proyecto, sin embargo, no cuenta con ningún puerto RJ-45 compatible con la tecnología PoE, por lo tanto, la solución planteada es utilizar un inyector PoE, el TL-PoE150S de TP-LINK (Figura 3.9). Este inyector se puede conectar a una fuente de tensión de 48V y a la red mediante un cable ethernet y combinarlos en otro puerto RJ-45 compatible con PoE, que es donde se conecta la cámara de Axis. De este modo, queda resuelto el problema de la alimentación de la cámara y la comunicación del robot con la misma.

Figura 3.9 Inyector PoE Tl-PoEE150S.

Convertidor DC/DC elevador

Finalmente, queda un último problema por resolver, ya que la tecnología PoE funciona con una tensión de 48V, pero la base del robot, donde se encuentran todos los puertos que ofrecen alimentación para los demás dispositivos, no cuenta con ningún puerto con una tensión tan alta. Sin embargo, debido a que el consumo máximo de la cámara es solo de 8W, se plantea conectar un convertidor de tensión tipo *boost* (Figura 3.10) al puerto libre de 12 V y la salida del mismo, de 48 V, al inyector PoE.

Figura 3.10 Convertidor DC/DC.

4 Software

Una vez establecido los componente físicos del sistema, se ofrece una descripción de las herramientas sobre las que se ha sustentado el desarrollo del software del robot.

4.1 ROS

ROS (Robotics Operative System) es un marco de aplicaciones *open source* ideado y mantenido por Open Robotics para desarrollar software en el campo de la robótica. Incluye librerías, herramientas y convenciones con el fin de simplificar las tareas más comunes en este entorno.

Al tratarse de un proyecto de código abierto, cuenta con una gran cantidad de recursos online para aprender a utilizarlo y de comunidades donde encontrar respuesta a problemas más concretos, lo que lo convierte en la herramienta ideal para el desarrollo de prototipos.

Para este caso, se utiliza la versión 16.04 Indigo, que fue lanzada en 2016 para Ubuntu 16.04 y aunque tanto esta versión de ROS como la de Ubuntu dejaron de recibir soporte y actualizaciones en 2020, se ha continuado el desarrollo de este robot con ellas con el objetivo de documentar lo mejor posible el proyecto antes de plantear una actualización de todo el sistema y evitar posibles incompatibilidades.

ROS es una herramienta con un gran potencial y que puede llegar a resultar muy complicada de entender en una primera aproximación, por lo que a continuación se expone una breve explicación, a nivel básico, de los conceptos más relevantes para este desarrollo: Los nodos, los mensajes y los servicios.

4.1.1 Nodos

Los nodos son los procesos que se llevan a cabo dentro del contexto de ROS. En esencia, son programas que se ejecutan en un ordenador, pero que pueden comunicarse entre ellos mediantes protocolos TCPROS, una adaptación del protocolo TCP/IP a las necesidades de ROS.

Sin embargo, el mayor potencial de esta herramienta reside en la versatilidad a la hora de lanzar estos nodos, ya que no necesitan ejecutarse en la misma máquina para compartir información entre ellos, sino que, como se ha apuntado anteriormente, utilizan un protocolo de la capa de aplicación para ello. De este modo, mientras la dirección IP de un nodo *maestro* sea conocida por todos los demás nodos, el ordenador de la red en el que se estén ejecutando será indiferente para ROS.

4.1.2 Mensajes

Es en este contexto donde cobran importancia los mensajes, estructuras de datos que utilizan los nodos para transmitir información a través del nodo *maestro*. Por otro lado, los *topics* proporcionan lo que podría entenderse como un canal de comunicaciones al que se envían mensajes y del que se pueden leer mensajes.

Por ejemplo, en la Figura 4.1 se representa al nodo *maestro* como ROS_master y a otros dos nodos como *node01* y *node02*. Además, existe un *topic* con el nombre *topic01*. En este caso, *node01* estaría publicando mensajes a *topic01* y tanto *ROS_master* como *node02* podrían recibir esa información. Del mismo modos, más nodos podrían publicar mensajes en el mismo *topic* y cualquier número de nodos podría leer de él.

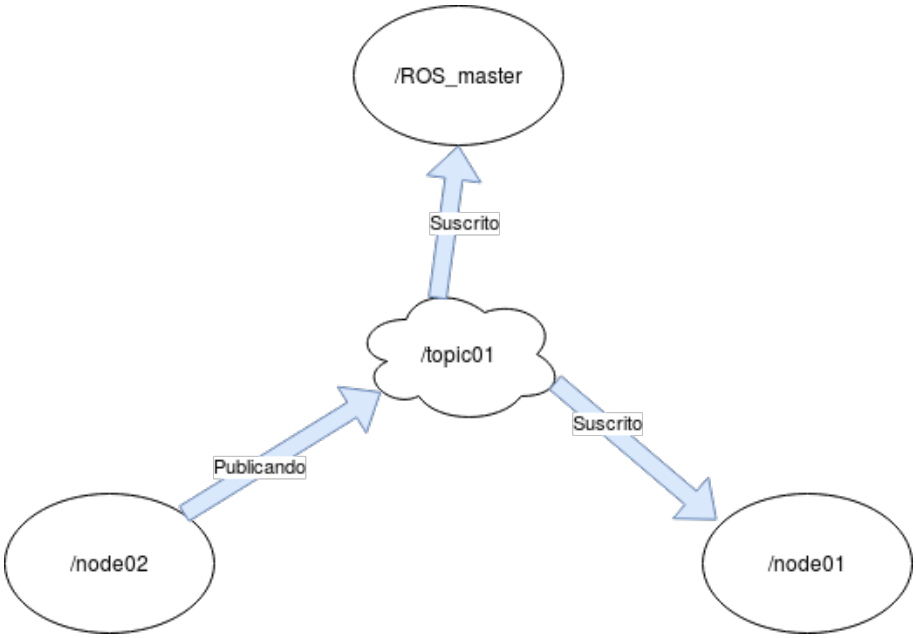


Figura 4.1 Estructura de nodos y mensajes.

4.1.3 Servicios

Por último, los servicios son tareas que se ejecutan como parte de un nodo cuando otro nodo lo solicita y emite una respuesta al terminar. Además, tanto dicha solicitud como la respuesta que emite el servicio al finalizar pueden transmitir información en una estructura de datos similar a la de los mensajes, permitiendo homogeneizar el tratamiento de los datos.

En la Figura 4.2 puede observarse como *ROS_master* cuenta con el servicio *Servicio01* y *node01* realiza peticiones a éste y recibe sus respuestas.

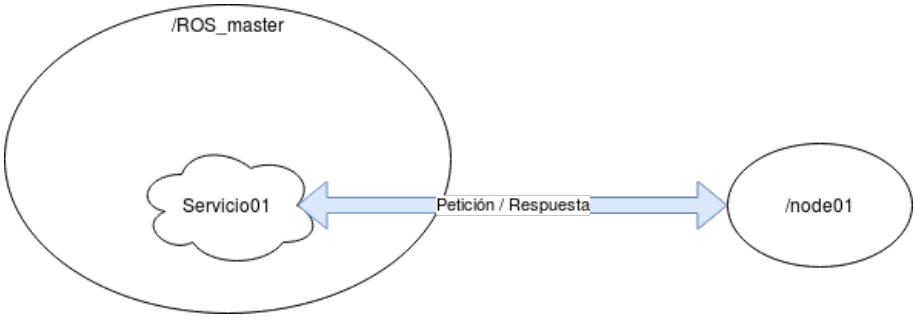


Figura 4.2 Estructura de nodos y servicios.

4.2 Gazebo

Como en muchos otros entornos, en el campo de la robótica la simulación es una parte crucial del proceso de desarrollo de nuevas tecnologías, ya que permite comprobar que los sistemas más básicos funcionan correctamente en contextos teóricos e ideales, antes de pasar a la fase de pruebas en el mundo real.

Para este fin se ha utilizado el software Gazebo (Figura 4.3), otra herramienta libre que permite realizar simulación del robot en un entorno físico aplicando los programas de ROS por lo que su implementación, en principio, es casi directa.

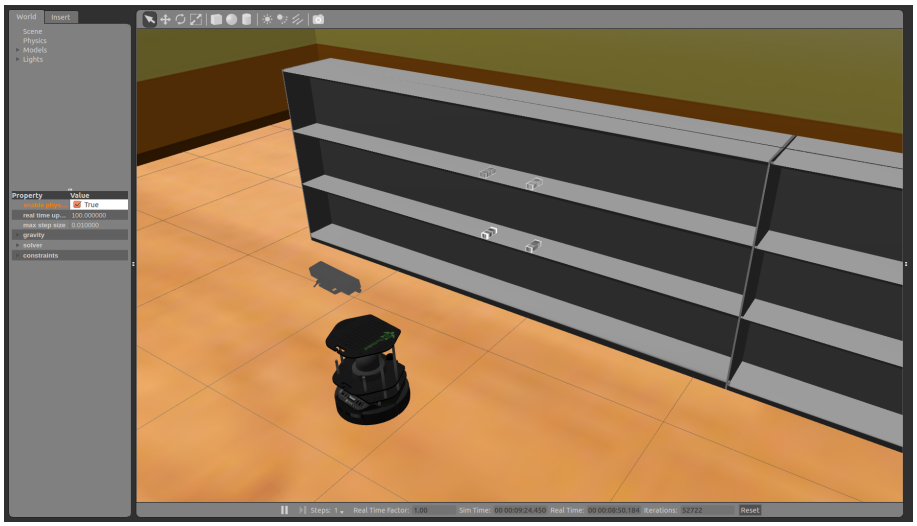


Figura 4.3 Entorno de simulación Gazebo.

4.3 Rviz

Rviz es una herramienta dentro de aplicaciones de ROS que ofrece la posibilidad de monitorizar de forma gráfica toda la información sensorial que recibe el robot, lo que la convierte en un recurso indispensable para realizar cualquier tipo de pruebas sobre el robot, tanto en entornos reales como en simulación, ya que puede utilizarse en combinación con Gazebo para detectar discrepancias entre el entorno y cómo "interpreta" el robot su entorno, ayudando a depurar posibles fallos en su programación.

Además, Rviz ofrece la posibilidad de personalizar su interfaz mediante QT, una conocida librería disponible para varios lenguajes de programación. Así, en trabajos anteriores sobre este proyecto, se desarrolló una interfaz de usuario mediante Rviz y QT, que es la que se sigue utilizando actualmente

4.4 Paquete Axis_node para ROS

Como se ha mencionado anteriormente, ROS es un conjunto de aplicaciones de software libre u *Open Source*, lo que generalmente significa que la comunidad en línea que lo utiliza a menudo publica herramientas y recursos bajo el mismo tipo de licencia, permitiendo que cualquier otra persona pueda utilizarla en sus propios proyectos.

En el caso de este trabajo, cabe resaltar el uso de un paquete de ROS desarrollado por Robotnik llamado *axis_node* y que ofrece una interfaz entre los servicios y mensajes de ROS y los protocolos de red que utiliza la cámara PTZ, así como un modelo de la misma para llevar a cabo simulaciones en Gazebo.

Por un lado, convierte los datos de imagen al formato de datos de los mensajes de ROS para poder acceder a ella desde otros nodos. Por otro lado, crea un nodo que ofrece varias posibilidades respecto al control de la cámara, por ejemplo, un servicio que la coloca en su posición inicial, o un *topic* al que se pueden enviar valores de *pan*, *tilt* y *zoom* para controlar la posición de la cámara.

De este modo, desde el punto de vista del ecosistema de ROS, la cámara PTZ no es más que otro nodo que interacciona con los demás a través de mensajes y servicios, aportando robustez y escalabilidad a todo el sistema.

Sin embargo, este controlador no permite acceder a todas las funcionalidades del dispositivos por lo que más adelante, al explicar la metodología de este trabajo, se especifican las modificaciones necesarias para acceder a las funciones más relevantes para este trabajo.

4.5 OpenCV

A la hora de llevar a cabo las tareas de procesamiento de imagen necesarias para realizar una primera aproximación a la localización de los códigos de barras, se ha optado por utilizar la librería OpenCV, un conjunto de funciones de código abierto para C++ y Python orientadas al procesamiento de imágenes y la visión artificial en tiempo real.

Para este caso, aunque la versión más actual es la 4.5.3 se ha utilizado la 2.4.8, por ser la más compatible con las versiones de ROS y Ubuntu con las que cuenta el sistema. A priori puede parecer que esto limitaría en gran medida las posibilidades del software, sin embargo, en este caso no se aplica ningún algoritmo de inteligencia artificial, que es hacia lo que evolucionan las nuevas versiones, por lo que los algoritmos más básicos de procesamiento de imágenes funcionan esencialmente igual.

4.6 Git

Como se ha mencionado anteriormente, la información sobre el proyecto se encontraba dispersa y escasamente documentada, además de tener muchos ficheros obsoletos que no habían sido eliminados.

Por otro lado, los sistemas de control de versiones son una herramienta básica en cualquier contexto de desarrollo software a día de hoy, ya que facilitan la organización y documentación del trabajo, especialmente cuando más de una persona participa en el mismo proyecto.

Así, se ha optado por integrar todos los recursos de software del sistema en un único repositorio en GitHub, facilitando el mantenimiento de una copia de seguridad actualizada y de un registro de todos los cambios realizados, en caso de que fuera necesario regresar a una versión anterior de la aplicación.

5 Metodología

Una vez establecidos los elementos físicos que componen el sistema y las herramientas de software sobre las que se desarrollaran los sistemas de control del robot, se plantean una serie de soluciones que permitan cumplir los objetivos propuestos anteriormente.

5.1 Planteamiento teórico

Antes de comenzar a plantear cualquiera de las propuestas, se hace imprescindible identificar en que punto del proceso interno de la aplicación del robot es necesario intervenir para evitar modificar más partes del proceso de las que son necesaria.

De este modo, se puede separar el funcionamiento del robot en tres fases: mapeado, localización de productos y guiado. Es en la segunda fase, a partir de ahora llamada simplemente localización para abreviar, donde se concentra el trabajo de esta intervención.

Anteriormente, en esta fase el robot ya dispone de un mapa de su entorno y cuenta con dos posibilidades, bien recibe ordenes directas sobre dónde debe buscar nuevos productos para escanear, o bien utiliza un algoritmo para identificar posibles estanterías y busca los productos de forma automática. Después, almacena en memoria la ubicación del producto en el mapa, la ubicación y orientación del robot cuando lo encontró y la información que estuviera codificada en su código QR.

Por lo tanto, y teniendo en cuenta que los objetivos principales de esta intervención incluyen leer códigos de barra con mayor precisión y traducir los códigos numéricos de los mismos a nombres de productos, se hace evidente que habrá que realizar modificaciones en el proceso para identificar los productos en el primer caso, y en el de almacenamiento de datos en el segundo.

Como primera aproximación teórica, se plantea un diagrama de flujo como el de la Figura 5.1, que comienza recibiendo una primera imagen del área a escanear y termina llamando a un servicio que gestiona el almacenamiento de los datos en la memoria del robot. Este servicio, llamado *BuscaProducto*, ha sido modificado en esta intervención para incluir una comunicación con un servidor remoto como se explica más adelante.

Entrando un poco más en detalle, el proceso comienza con una cámara, bien la Kinect o bien la Axis, tomando una imagen a color de la zona donde el robot intente encontrar códigos de barras. Esta imagen es después procesada para delimitar áreas de la misma donde pueden encontrarse códigos de barras y se calcula la posición en el espacio que ocupan estos códigos. A continuación, se entra en un bucle en el que la cámara PTZ se orienta hacia los códigos detectados uno a uno y aumenta el zoom para poder interpretarlos correctamente. Cuando ya ha pasado por todos los potenciales códigos se pasan los valores numéricos obtenidos de los mismo por el servidor remoto y se almacenan los datos en la memoria del robot. Este proceso se lleva a cabo cada vez que el robot intenta escanear una nueva zona con posibles códigos de barras.

A la hora de implementarlo en el ecosistema de ROS, se plantea el desarrollo de dos nodos como base para el proceso de escaneo de códigos y orientación de la cámara, que se llamarán *barcode_scan* y *barcode_manager* respectivamente.

Cabe destacar que, para los procesos del cálculo de la posición de los códigos y la orientación de la cámara PTZ se han desarrollado dos propuestas distintas en las que se profundizará más adelante.

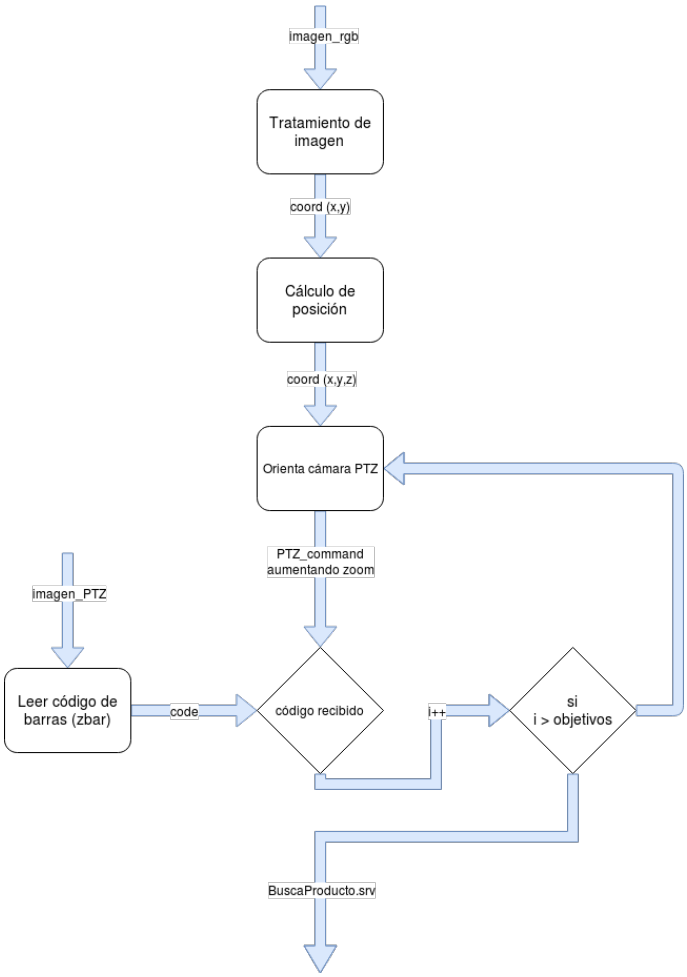


Figura 5.1 Diagrama de flujo de la nueva funcionalidad.

5.2 Procesamiento de imagen (*barcode_scan*)

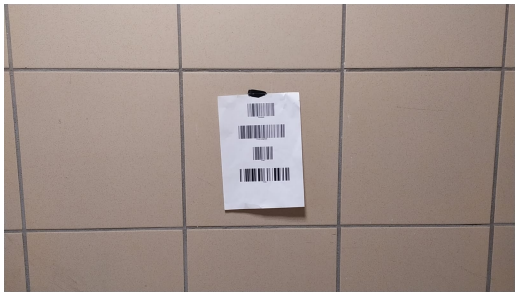
Para, a través de una sola imagen, identificar áreas donde pueden existir códigos de barras hacia los que orientar la cámara PTZ para aumentar la nitidez, se ha desarrollado un algoritmo de procesamiento de imágenes basado en el filtro Scharr, que consiste en aplicar una derivada discreta en los eje x horizontal y vertical de la imagen. En este caso, solo se aplica el filtro en el eje horizontal, ya que se considera que todos los códigos de barras estarán colocados en estanterías con sus barras en forma vertical. De este modo, al aplicar la derivada parcial sobre el eje x de la imagen en blanco y negro, destacarán los códigos de barras.

Sin embargo, el algoritmo no se limita a aplicar dicho filtro. Para empezar, cada vez que se captura una nueva imagen (5.2a) desde la cámara, se transforma a una imagen en blanco y negro para facilitar el resto del procesamiento (5.2b).

A continuación, se aplica un filtro Scharr (Figura 5.3) en el eje x, que permite resaltar las zonas donde hay un alto contraste en un eje determinado, en este caso el horizontal, aplicando una derivada parcial a la imagen.

Después, se suaviza la imagen y se le aplica un filtro umbral (Figura 5.4) para limpiarla del ruido que haya podido ocasionar el filtro anterior.

Para aislar las zonas relevantes, se aplican un filtro de erosión y dilatación y, sobre este, un algoritmo de búsqueda de contornos, que permite modelar como rectángulos las zonas que hayan quedado marcadas después de todo este proceso, obteniendo el centro y el tamaño de cada potencial código de barras como se muestra en la Figura 5.5.



(a) Imagen original.

(b) Imagen en blanco y negro.

Figura 5.2 Primer filtrado de la imagen.

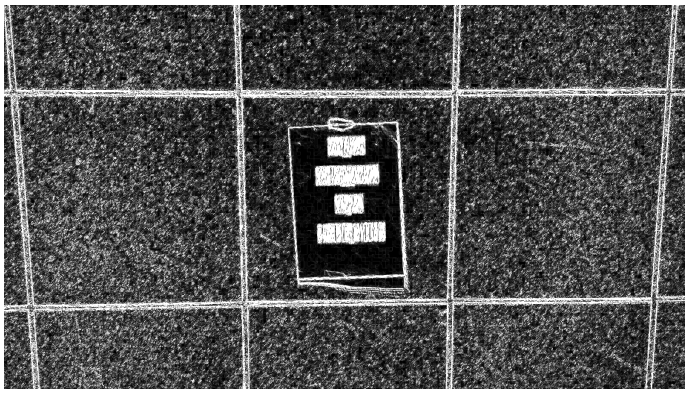


Figura 5.3 Filtro Scharr.

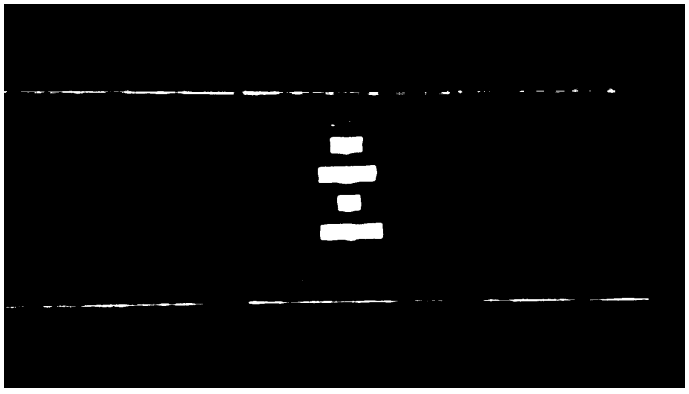
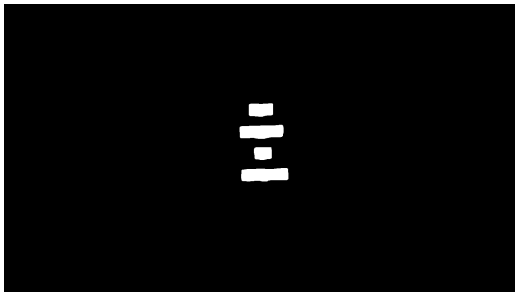
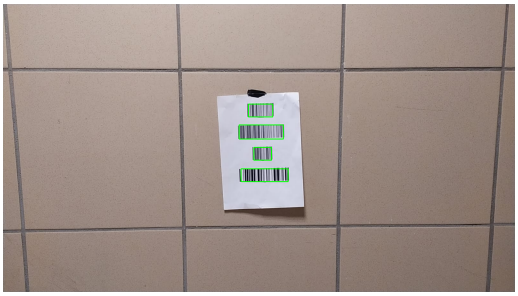


Figura 5.4 Filtro umbral.



(a) Dilatación y erosión.



(b) Códigos de barras detectados.

Figura 5.5 Filtro final.

5.3 Localización de productos en el espacio barcode_manager

A la hora de idear un algoritmo que permita ubicar los códigos de barras en el espacio antes de poder interpretarlos, se plantean dos alternativas.

Por un lado, la cámara Kinect que se venía utilizando en versiones anteriores ofrece la posibilidad de tomar imágenes de profundidad a través de un sensor infrarrojo y ya se había desarrollado un proceso para estimar la posición en el espacio de los códigos QR recibidos [6], por lo que resultaba relativamente sencillo aprovechar ese trabajo y construir sobre él la programación de la cámara PTZ.

Por otro lado, gracias a una funcionalidad con la que cuenta el software interno de la cámara Axis 5054, es posible indicarle a la cámara PTZ una región del espacio sobre la que encuadrar la imagen, por lo que una vez obtenidas las coordenadas bidimensionales de los códigos de barras, la orientación de la cámara es inmediata. En este caso, el uso de la cámara Kinect ya no sería necesario, disminuyendo tanto el tamaño como el peso del robot y aumentando su autonomía, además de simplificar su programación.

5.3.1 Primera propuesta: Cámara de profundidad y cámara PTZ

Esta propuesta plantea el tanto de la cámara Kinect como la PTZ en paralelo, aprovechando por un lado el sensor de profundidad de una y el zoom óptico de la otra. Así, si se observa de nuevo la Figura 5.1, esta solución trataría los procesos de *cálculo de posición* y *orientación de la cámara PTZ*

5.3.2 Cálculo de posición

Para realizar una estimación de coordenadas tridimensionales de los códigos de barras, se toma en primer lugar la información generada por el algoritmo de procesamiento de imagen realizado previamente, en este caso, a partir de una imagen a color tomada por la cámara Kinect. Este algoritmo proporciona una estructura de datos que contiene las coordenadas sobre la imagen, es decir, en dos dimensiones, de las zonas donde pueden encontrarse códigos de barras.

Al mismo tiempo, se toma la nube de puntos generada por el sensor de profundidad de la cámara Kinect, que proporciona información sobre la ubicación en el espacio de los objetos frente a la cámara.

Ambos resultados son combinados, tomando las coordenadas de los puntos de la nube de puntos que corresponden con los píxeles de los potenciales códigos de barras.

Esta nueva estructura de datos, que cuenta con una serie de coordenadas tridimensionales, a un nodo que se encarga de gestionar la orientación y zoom de la cámara PTZ cuando se solicita mediante un servicio de ROS.

Orientación de la cámara PTZ

Para comprender la metodología seguida en este apartado, es necesario recordar que el controlador utilizado para la comunicación con la cámara PTZ, *Axis_node*, solo puede modificar la orientación de la misma cuando recibe mensajes de ROS indicando los ángulos en radianes de los dos ejes de rotación y el zoom tomando valores entre 1 y 9999. Por lo tanto, es necesario calcular la combinación de ángulos y zoom a partir de las coordenadas tridimensionales de los códigos obtenidas previamente.

Para ello, primero se realiza un cambio de base sobre todos los puntos con potenciales códigos de barras, de modo que la base de coordenadas pase de ser la cámara Kinect a la cámara PTZ. Esto es posible gracias una de las librerías de ROS y al modelo del robot que se utiliza en simulación y que tiene las mismas dimensiones que el robot real.

Después, para obtener los valores de los tres grados de libertad de la cámara para cada uno de los puntos, se calculan las ecuaciones del modelo cinemático inverso de la cámara mediante el proceso de Denavit-Hartemberg [10], un modelo matemático muy utilizado en robots manipuladores que permite obtener la posición de las articulaciones a partir del punto que ocupa en el espacio el efector final de un brazo robótico. Para este caso, se ha considerado la cámara PTZ como un brazo de tres grados de libertad, con dos articulaciones de rotación (*pan* y *tilt*) y una tercera prismática (zoom) obteniendo las siguientes ecuaciones:

$$pan = tan^{-1}\left(\frac{p_x}{p_y}\right)$$

$$tilt = tan^{-1}\left(\frac{p_z}{\sqrt{p_x^2 + p_y^2}}\right)$$

Para el valor del zoom se realizar una estimación más sencilla, ya que requiere de una precisión que no se ha logrado alcanzar en las pruebas en un entorno real, por lo que, cada vez que se oriente la cámara hacia un nuevo objetivo, se aumenta el zoom progresivamente hasta que se detecte algún código o se alcance el valor máximo.

Simulación

Antes de implementar la solución en el robot real, se realizaron varias pruebas en el entorno de simulación Gazebo (Figura 5.6) con códigos de barras de 10 cm de ancho. Al mismo tiempo, a través de Rviz (Figura 5.8), se podía comprobar como el algoritmo colocaba los marcadores en los lugares correspondientes a los códigos.

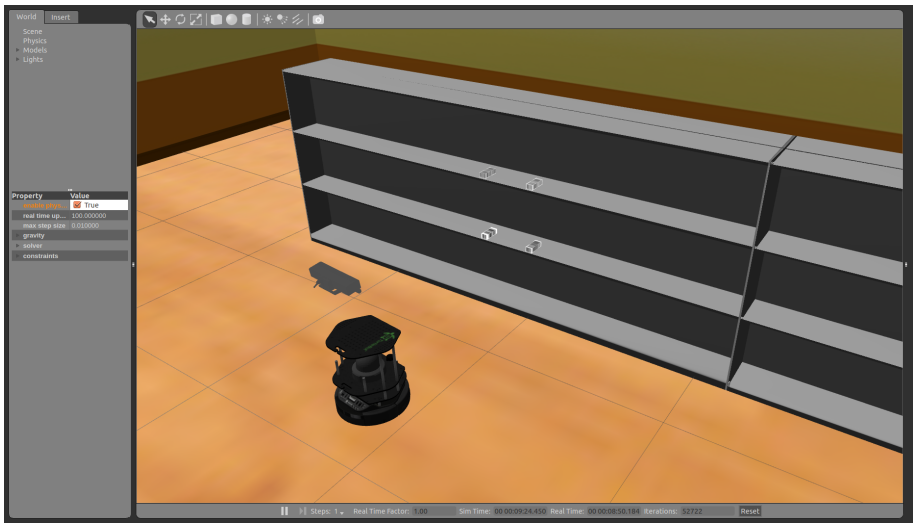


Figura 5.6 Entorno de simulación en Gazebo.

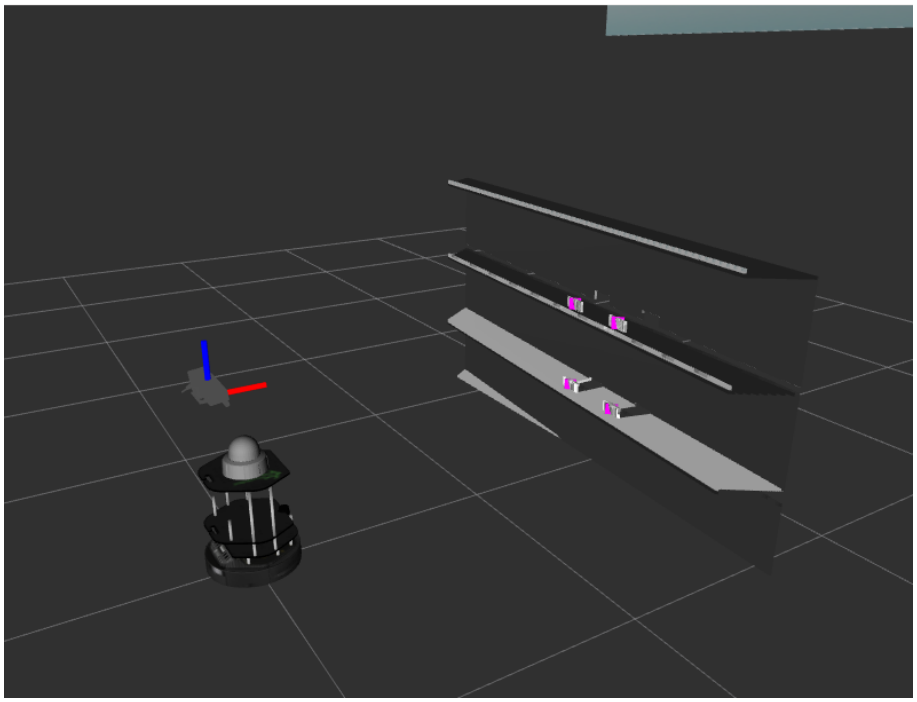


Figura 5.7 Posicionamiento de los códigos de barras (marcadores rosas).

Por otro lado, para simular el comportamiento del zoom óptico es necesario tratar la cámara como un robot de tres grados de libertad, de forma similar a como se calcularon las ecuaciones del modelo cinemático inverso, tomando las imágenes desde el extremo de la articulación prismática. Esta aproximación obvia las

distorsiones asociadas a la lente al variar la distancia focal, pero dentro de Gazebo no existen a día de hoy opciones para la simulación de cámaras con una distancia focal variable.

En cualquier caso, el resultado dentro de las simulaciones es lo suficientemente bueno como para trasladarlo al sistema real.

Figura 5.8 Orientación y zoom de la cámara PTZ en Gazebo.

5.3.3 Segunda propuesta: Cámara PTZ

Para esta solución, se decide aprovechar una funcionalidad integrada en el software que tiene de serie la cámara Axis M5054 pero que, en principio, no es accesible desde el controlador de *Axis_node*, pero que permite indicarle al software de la cámara las coordenadas de la imagen sobre las que se desea encuadrar y cuanto se desea aumentar el tamaño aparente de los objetos que en ella aparecen, con un máximo de 5 aumentos para el zoom óptico.

Modificación sobre *Axis_node*

El paquete de ROS *Axis_node* proporciona, entre otras cosas, un nodo escrito en Python que está suscrito a varios *topics* y crea servicios que pueden ser llamados por otros nodos. Para enviar comandos a la cámara PTZ o recibir información sobre esta, utiliza mensajes HTTP del tipo GET REQUEST o SET REQUEST con los parámetros correspondientes.

Por lo tanto, para incluir en el nodo existente la capacidad de acceder a otra funcionalidad de la cámara solo es necesario conocer la URL correspondiente a ese comando y los parámetros necesarios. Esto se ha conseguido gracias a la aplicación web que trae la cámara Axis M5054, que permite manejar todas las herramientas de la cámara y analizar cómo funcionan.

Así, se completa esta modificación creando un nuevo servicio dentro de este nodo llamado *focus_PTZ*, cuya función correspondiente es *focusService*, que puede ser llamado desde cualquier otro nodo y que recibe como parámetros el tamaño de la imagen en píxeles, la rotación que se quiere aplicar sobre la misma (ninguna en este caso), las coordenadas donde se encuentra el objeto a centrar y el zoom. Después simplemente se envía una petición HTTP SET y se espera que la cámara devuelva el código 204, indicando que la solicitud se ha recibido con éxito.

```
        def focusService(self, req):
        imagewidth = req.imagewidth
        imageheight = req.imageheight
        imagerotation = req.imagerotation
        x_coord = req.x_coord
        y_coord = req.y_coord
        zoom = req.zoom
        areazoom = str(x_coord) + "," + str(y_coord) + "," + str(zoom)
        params = {'areazoom':areazoom, 'imagewidth':imagewidth, 'imageheight':
            imageheight, 'imagerotation':imagerotation}
        url = "/axis-cgi/com/ptz.cgi?camera=1&%s" % urllib.urlencode(params)
        conn = httplib.HTTPConnection(self.hostname)

        try:
            url = "/axis-cgi/com/ptz.cgi?camera=1&%s" % urllib.urlencode(params)
            conn.request("GET", url)
            if conn.getresponse().status != 204:
```

```
                rospy.logerr('%s/sendPTZCommand: Error getting response. url = %
                    s%s' %
                            (rospy.get_name(), self.hostname, url))
        except socket.error, e:
            rospy.logerr('%s:sendPTZCommand: error connecting the camera: %s ' %
                 (rospy.get_name(), e))
        except socket.timeout, e:
            rospy.logerr('%s:sendPTZCommand: error connecting the camera: %s ' %
                 (rospy.get_name(), e))

        return {}
```

Orientación de la cámara PTZ

En este caso, al contar con un servicio de ROS que gestiona el control de la orientación de la cámara ya no es necesario utilizar ningún modelo cinemático inverso, ya que la cámara PTZ es capaz de posicionarse solo conociendo la región de la imagen que debe encuadrar.

Sin embargo, al no contar con el sensor de profundidad de la cámara Kinect, es necesario realizar una estimación diferente para localizar en el espacio los productos detectados por la cámara PTZ.

Como se ha mencionado anteriormente, el nodo que controla la cámara permite enviarle instrucciones a ésta, pero también recibir información de la misma.

Por lo tanto, se puede suponer una base de coordenadas como la de las figuras 5.9 y 5.10, donde los ángulos *pan* y *tilt* son obtenidos a través de una petición HTTP GET y la distancia en el eje X respecto a la estantería donde estaría el producto se obtiene de los datos proporcionados por el sensor lidar. De este modo, se pueden calcular las coordenadas Y y Z aplicando unas relaciones trigonométricas sencillas:

$$p_y = -tan\,(pan) * p_x$$
$$p_z = tan\,(tilt) * p_x$$

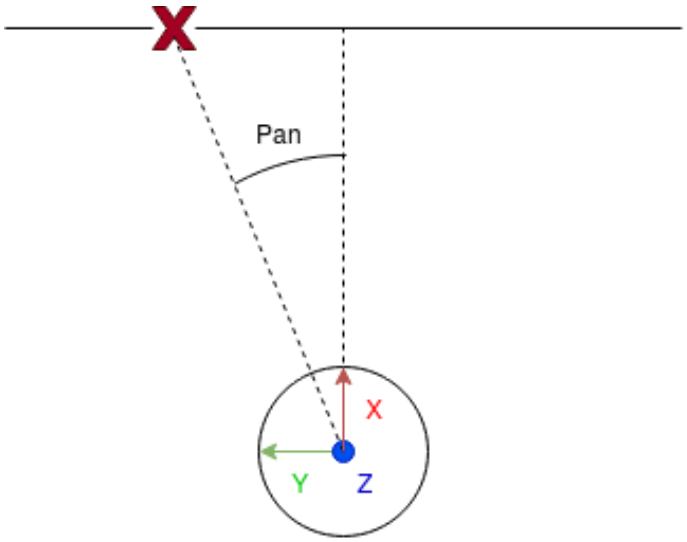


Figura 5.9 Calculo de la posición a través del ángulo *pan*.

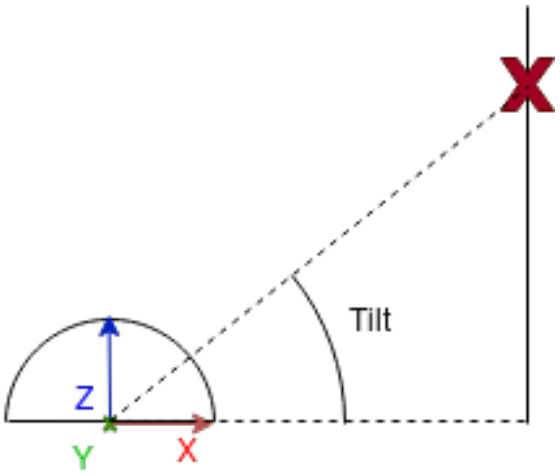


Figura 5.10 Calculo de la posición a través del ángulo *tilt*.

Simulación

Al haber comprobado en simulación gran parte del sistema para la prueba anterior, y teniendo en cuenta que no es posible simular en gazebo la conexión entre ROS y el software de la cámara, las pruebas para esta propuesta se llevan a cabo en el robot real, comprobando que efectivamente la cámara PTZ es capaz de encuadrar las áreas donde se encuentran potenciales códigos de barras e interpretarlos (Figuras 5.11 y 5.12).

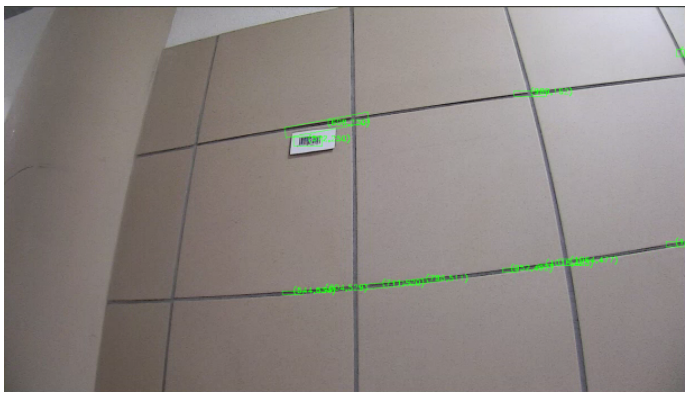


Figura 5.11 Detección de posibles códigos de barras.

Figura 5.12 Lectura del código de barras.

5.4 Comunicación con el servidor remoto

Finalmente, una vez que se han obtenido la ubicación y el número de cada código de barras visible, se establece una conexión con un servidor remoto para obtener el nombre del producto correspondiente, para lo que es necesario que la red WiFi a la que se conecta el robot tenga acceso a internet.

Para gestionar esta conexión, primero se ha creado un nodo en Python que ofrece un servicio al cual se le puede enviar el número de un código de barras. Este servicio envía ese número como una petición a una API externa que, cuando recibe un valor numérico correcto, devuelve una estructura de datos en formato JSON. Para esta aplicación, sin embargo, solo es necesario el campo correspondiente al nombre.

A continuación se puede leer el código correspondiente a este servicio, que devuelve como respuesta del servicio la cadena de caracteres correspondiente al nombre del producto solicitado.

```
from __future__ import print_function
from bbdd_productos.srv import *
import rospy
import requests
import json

parameters = {"apiKey":"dKq1COMWXaQZ1jYja09nRhKUpLsKlN", "uidActividad":"FOOD"}
headers = {"Accept":"application/json"} # Formatea la respuesta de la API a
    json

def handle_API_request(req):
    url = "http://demo.comerzzia.com/comerzzia/ws/articulos/ean/" + req.data
    response = requests.get(url, params=parameters, headers=headers)
    if (response.status_code==200):
        print(response.json()[0]['desArticulo'])
        return response.json()[0]['desArticulo']
    else:
      print("Error: "+str(response.status_code))
      return str(response.status_code)
    #print("Returning [%s]"%(req.data))

def API_request_server():
    rospy.init_node('API_request')
    s = rospy.Service('API_request', APIProducto, handle_API_request)
    print("Ready to send request.")
    rospy.spin()

if __name__ == "__main__":
    API_request_server()
```

Por último, se modifica ligeramente el nodo *movimiento_nodo*[6], incorporando una llamada al servicio que realiza las consultas al servidor externo, de modo que el robot almacene en su memoria interna los nombres de los productos en lugar de sus códigos numéricos.

```
for (int i = 0; i < codigos.size(); ++i)
{
    apiprod.request.data = codigos[i];
    convertCode.call(apiprod);
    if(apiprod.response.data == "404") // Si no está en la API, lo elimina (se
        considera un error de lectura)
    {
        codigos.erase(codigos.begin() + i);
        tipos.erase(tipos.begin() + i);
        map_x.erase(map_x.begin() + i);
        map_y.erase(map_y.begin() + i);
        map_z.erase(map_z.begin() + i);
        base_x.erase(base_x.begin() + i);
        base_y.erase(base_y.begin() + i);
        base_z.erase(base_z.begin() + i);
        i--; // Si se borra, se vuelve a iterar en la misma componente que toma
             su lugar
    }else{ // Si no hay error, le asigna el nombre proporcionado
        codigos[i] = apiprod.response.data;
    }
}
```

6 Análisis de resultados

Para llevar a cabo un análisis de los resultados obtenidos en este trabajo, es necesario remitirse a los objetivos principales planteados inicialmente:

- Incluir la capacidad de leer códigos de barras con la misma facilidad que códigos QR en la fase de localización de productos.
- Establecer una comunicación entre el robot y la base de datos del comercio para obtener los nombres de los productos a partir de su código numérico.

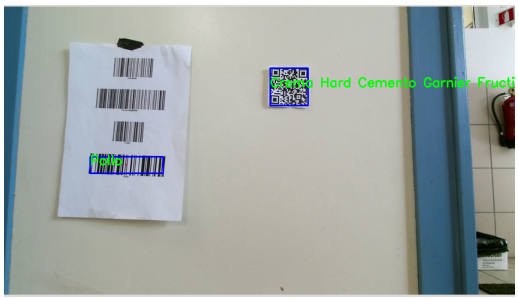
Para determinar hasta que punto se ha cumplido el primer objetivo, es necesario comparar con una mínima exactitud el punto de partida del sistema y hasta donde llegan sus capacidades tras la intervención. Con este fin, se establece un escenario de pruebas con códigos de barras de entre 8 y 16 cm de ancho y códigos QR de tamaños similares. A continuación, se reduce progresivamente la distancia entre el robot y los códigos, tomando nota de cuando empiezan a ser legibles por la cámara.

(a) Imagen tomada por la cámara Kinect. **(b)** Robot frente al objetivo.

Figura 6.1 Prueba a 160 cm del objetivo.

(a) Imagen tomada por la cámara Kinect. **(b)** Robot frente al objetivo.

Figura 6.2 Prueba a 76 cm del objetivo.



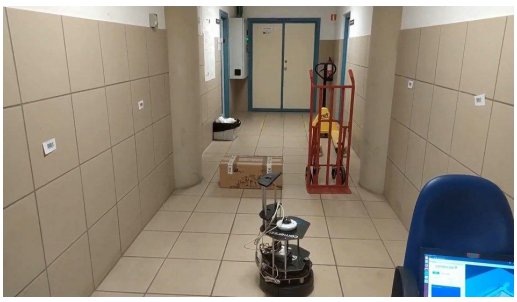
(a) Imagen tomada por la cámara Kinect.

(b) Robot frente al objetivo.

Figura 6.3 Prueba a 40 cm del objetivo.

Como puede observarse especialmente en la Figura 6.3 la distancia mínima a la que el robot puede empezar a interpretar los códigos más grandes es relativamente pequeña, solo 40cm. En comparación, en la Figura 6.4 se puede observar al robot a una distancia e la pared de 1 metro capturando con nitidez un código de 10 cm de ancho a una altura de 1.5 m, lo que puede considerarse como una mejora considerable en las capacidades del sistema.

(a) Imagen tomada por la cámara PTZ.



(b) Robot frente al objetivo.

Figura 6.4 Prueba a 1 m del objetivo.

En cuanto al establecimiento de la comunicación con la base de datos del comercio, resulta más sencillo evaluar los resultados, ya que se puede comprobar su correcto funcionamiento simplemente observando que, al terminar la etapa de localización de productos, el listado de los productos disponible contiene sus nombres comerciales en lugar del valor numérico del código de barras (Figura 6.5)

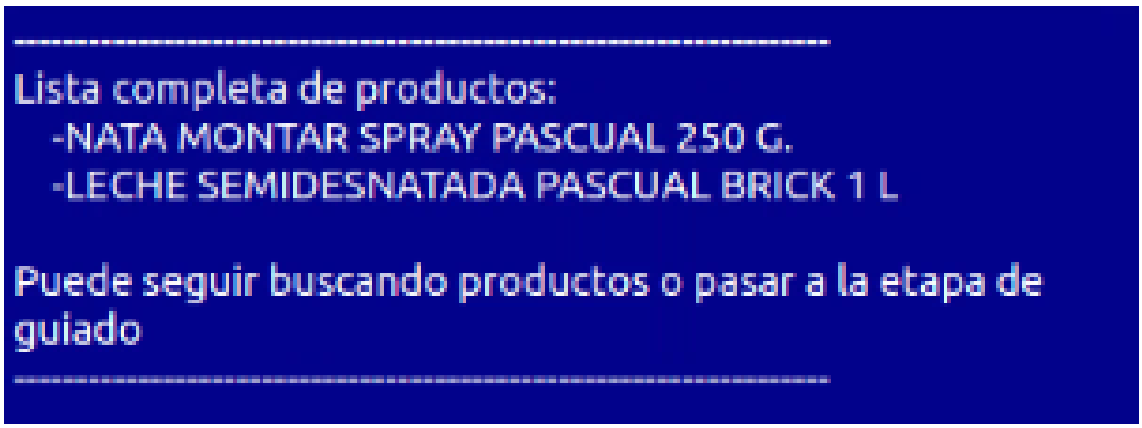


Figura 6.5 Listado de productos disponibles.

6.1 Comparación de ambas propuestas

Si bien es cierto las dos propuestas planteadas para resolver el problema del control de la cámara PTZ ofrecen una precisión similar en sus resultados, existen otros puntos a tener en cuenta para desechar una de ellas en favor de la otra.

En primer lugar, la primera propuesta presenta un consumo energético mayor que la segunda, ya que tiene que mantener en funcionamiento un dispositivo extra que, a pesar de no necesitar de una gran potencia eléctrica, conlleva una disminución de la autonomía del robot.

En segundo lugar, desde el ecosistema de ROS resulta mucho más sencillos utilizar un controlador que base su funcionamiento en mensajes del protocolo HTTP dirigidos a una dirección IP que comunicarse con un periférico a través del puerto serie. Además, el controlador de la cámara Kinect no ha sido desarrollado por su fabricante, sino que es el resultado de un proceso de ingeniería inversa que en ocasiones pierde la conexión con el periférico sin motivo aparente.

En conclusión, eliminar la cámara Kinect del sistema, manteniendo el sensor lidar como detector de obstáculos, puede suponer una mejora tanto en la autonomía del robot como en en su fiabilidad y escalabilidad, lo que convierte a la segunda propuesta en la solución definitiva.

7 Conclusion

A lo largo del desarrollo de este proyecto, se han presentado diversos problemas y dificultades que han abarcado temas muy diversos, lo cual es de esperar una vez que se comprende la gran cantidad de procesos que se llevan a cabo para tareas aparentemente tan sencillas como encuadrar una región de una imagen más grande.

Este trabajo, sin duda, ha puesto de manifiesto el carácter interdisciplinar de los proyectos de robótica ya que, aunque en este caso no se haya profundizado en gran medida en algoritmos de visión artificial, modelos cinemáticos, protocolos de comunicación o electrónica, ha sido necesario integrar todos esos campos en un solo sistema.

En cuanto al desarrollo a nivel personal que ha supuesto la elaboración de este trabajo, más allá de los cocimientos técnicos aprendidos y aplicados, caben destacar las habilidades adquiridas en materia de análisis de problemas reales, planteamiento de soluciones y planificación del trabajo, así como la comunicación de ideas abstractas para establecer consensos con la empresa contratante.

7.1 Nuevas lineas de trabajo y posibles mejoras

Por último, para finalizar la exposición de este trabajo, se plantean una serie de lineas de trabajo con vistas a futuro para continuar mejorando las funcionalidades del sistema e incluir algunas nuevas.

7.1.1 Algoritmo de inteligencia artificial

Actualmente, el algoritmo de procesamiento de imágenes que se utiliza para detectar posibles códigos de barras es un proceso extremadamente simple que necesita de un entorno muy controlado para funcionar correctamente, ya que a menudo confunde sombras con códigos de barras y puede retrasar enormemente la etapa de localización de productos.

Por lo tanto, se propone la sustitución del algoritmo existente por algún modelo de detección de objetos desarrollado con este fin, o el uso de una API de visión artificial como la que oferta Google, aunque en ese caso es necesario pagar una suscripción y mantener una conexión estable a internet durante todo el proceso

7.1.2 Actualizar el sistema a versiones más recientes

En el momento de redactar este trabajo, la mayor parte del software que utiliza el robot cuenta con versiones obsoletas que en el futuro pueden comprometer la seguridad de la información del usuario, por lo que se propone actualizar, al menos, la versión del Sistema Operativo y la de ROS, ya que versión Noetic de esté mismo es la primera que será mantenida durante un periodo mayor a un año, por lo que probablemente termine siendo la más estable hasta la fecha.

Por otro lado, la última versión de ROS también es compatible con las versiones más nuevas de OpenCV, lo cual podría facilitar el desarrollo de las mejoras propuestas en el punto anterior.

7.1.3 Interfaz de usuario a través de una aplicación web

Por último, no hay que olvidar que este sistema se desarrolla con el fin de ser utilizado en un entorno comercial, donde el usuario final no tiene que tener conocimientos de ingeniería. En este caso, ya existe una interfaz

de usuario basada en Rviz, sin embargo, es necesario acceder a ella a través de un ordenador que ya tenga instalado ROS y varias librerías adicionales.

Una solución más eficaz podría ser desarrollar una aplicación web que funcione localmente en el robot y a la que cualquier usuario pueda conectarse desde cualquier dispositivo fácilmente para controlar y monitorizar al robot.

Apéndice A
Manual de instalación

A continuación se detalla el proceso de instalación del Software del proyecto tanto para el ordenador del robot o *TurtleBot* como para el ordenador convencional que hará las veces de estación de desarrollo y monitorización.

Para todo el proceso, se supone que ambos ordenadores tienen instalados previamente Ubuntu 16.04 como su sistema operativo, bien como único SO o como una partición del disco duro, y Python 2.7.

A.1 Instalación de Git

Para instalar Git, el procedimiento es bastante similar al de casi cualquier otro software en Ubuntu. Basta con abrir un terminal del linux e introducir los siguientes comandos:

```
sudo apt-get update
sudo apt-get install git
```

A.2 Instalación de ROS Indigo

El proceso para instalar ROS Indigo, que es la versión destinada a Ubuntu 16, es algo más complejo.

A.2.1 Configurar los repositorios de Ubuntu

Para ello hay que abrir el Centro de Software de Ubuntu y en el menú desplegable Editar seleccionar Orígenes de Software. Después, marcar las opciones con las etiquetas *restricted*, "universe" y "multiverse".

A.2.2 Permisos de los paquetes de ROS

Después, para permitir que el ordenador acepte los paquetes de ROS, se ejecutan los siguientes comandos desde un terminal:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
    main" > /etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-key 0xB01
    FA116

sudo apt-get update
```

A.2.3 Instalar ROS

A continuación, para instalar ROS al completo, se instalará la versión de escritorio con el siguiente comando:

```
sudo apt-get install ros-indigo-desktop-full
```

A.2.4 Variables de entorno

Para configurar las variables de entorno de ROS, se pueden utilizar estos dos comandos:

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc
echo "source /opt/ros/indigo/setup.bash" >> ~/.profile
source ~/.profile
```

En caso de usar un cliente para el terminal distinto a bash como ohmyzsh, habrá que realizar un proceso similar sobre el archivo rc correspondiente.

Después de ejecutar los comandos, es necesario reiniciar el terminal para aplicar los cambios.

A.2.5 Instalar paquetes de ROS

Este proyecto depende de múltiples librerías de ROS para funcionar correctamente. A continuación se muestra un comando de instalación que los une todos. En caso de error, es recomendable instalarlos uno a uno por orden para para hallar la fuente del mismo más fácilmente.

```
sudo apt-get install ros-indigo-turtlebot ros-indigo-turtlebot-apps ros-indigo-
    turtlebot-
interactions ros-indigo-turtlebot-simulator ros-indigo-kobuki-ftdi ros-indigo-
    rocon-remocon
ros-indigo-rocon-qt-library ros-indigo-ar-track-alvar-msgs ros-indigo-joy ros-
    indigo-object-
recognition-msgs ros-indigo-zbar-ros ros-indigo-controller-manager ros-indigo-
    costmap-2d
ros-indigo-kobuki-msgs ros-indigo-nav2d ros-indigo-nav-core ros-indigo-navfn
    ros-indigo-
people-tracking-filter ros-indigo-yocs-velocity-smoother ros-indigo-kobuki-
    safety-controller
ros-indigo-smart-battery-msgs ros-indigo-move-base-msgs ros-indigo-social-
    navigation-layers
ros-indigo-joint-trajectory-controller ros-indigo-velocity-controllers ros-
    indigo-hokuyo-node ros-indigo-position-controllers ros-indigo-effort-
    controllers ros-indigo-joint-state-controllers ros-indigo-gazebo-ros-pkgs
```

Por último, para configurar la conexión del ordenador con la base de Kobuki, es necesario abrir dos terminales. En el primero, ejecutar el comando:

```
roscore
```

En el segundo:

```
source /opt/ros/indigo/setup.bash
rosrun kobuki_ftdi create_udev_rules
```

A.2.6 Clonación del repositorio

Tras instalar Git y Ubuntu, es hora de descargar los archivos del proyecto. Para ello, primero hay que crear un espacio de trabajo para ROS, ejecutando los siguientes comandos en la ubicación donde se deseen descargar.

```
mkdir -p catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

Después se ejecuta una primera compilación:

```
cd ~/catkin_ws
catkin_make
```

Ahora ya se puede clonar el repositorio de git en el espacio de trabajo. Para ello, suponiendo que el sistema de archivos que se ha seguido en este apéndice, se ejecutan los siguientes comandos en el terminal:

```
cd ~/catkin_ws/src
sudo rm CMakeLists.txt
sudo git clone https://github.com/pedmacros/turtlebot-tier.git
```

Al no tratarse de un repositorio público, se pedirán al usuario las claves de acceso y después comenzará la descarga.

A.2.7 Compilación del *Workspace*

A continuación, se pueden compilar los paquetes siguiendo esta secuencia de comandos. Se recomienda seguir el mismo orden, ya que algunos paquetes dependen de los ficheros compilados de otros.

```
catkin_make --pkg catkin_simple
catkin_make --pkg rosbridge_suite
catkin_make --pkg turtlebot
catkin_make --pkg zbar_ros
catkin_make --pkg position_detector
catkin_make --pkg pcl_recognition
catkin_make --pkg lanzadores
catkin_make --pkg using_markers
catkin_make --pkg nav2d
catkin_make --pkg nav2d_navigator
catkin_make --pkg bbdd_productos
catkin_make --pkg sos
catkin_make --pkg etapa_guiado
catkin_make --pkg localizacion_estanterias
catkin_make --pkg localizacion_productos
catkin_make -- pkg programa_central
catkin_make --pkg programa_central_turtlebot
catkin_make --pkg bateria
catkin_make --pkg rviz_plugin_tutorials
catkin_make
```

A.3 Instalación de OpenCV 2.4.9

Para la instalar la versión 2.4.9 de OpenCV, es necesario empezar instalando varias dependencias:

```
sudo apt-get install build-essential cmake pkg-config
sudo apt-get install libjpeg62-dev libtiff4-dev libjasper-dev
sudo apt-get install libgtk2.0-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
```

A continuación, se puede descargar OpenCV 2.4.9 de *https://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.9/*. Es necesario descomprimir el fichero en la ubicación donde se desee realizar la instalación. Se recomienda crear un directorio para ello en *home*. Después, se crea un directorio *build* dentro del directorio producto del fichero comprimido de OpenCV. Desde ahí, se puede comenzar la instalación con el siguiente comando:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE -D MAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB
    =ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON
    -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_
    OPENGL=ON ..
```

Después, ejecutar:

```
make
```

Y, por último:

```
sudo make install
```

A.4 Kinect v2 libfreenect2

Para que los paquetes de ROS que se comunican con la cámara Kienct V2 funcionen correctamente, es necesario instalar libfreenect2, un controlador de software libre para dicha cámara.

Para empezar, se clona el repositorio:

```
git clone https://github.com/OpenKinect/libfreenect2.git
cd libfreenect2
```

Después, si no se ha hecho previamente, se instalan las herramientas de compilación:

```
sudo apt-get install build-essential cmake pkg-config
```

A continuación, se instalan varias librerías de las que depende el controlador:

```
sudo apt-get install libusb-1.0-0-dev
sudo apt-get install libturbojpeg libjpeg-turbo8-dev
sudo apt-get install libglfw3-dev
```

Por último, se compilan los ficheros con:

```
cd ~/libfreenect2
mkdir build && cd build
cmake .. -DENABLE_CXX11=ON i -DCMAKE_INSTALL_PREFIX=$HOME/freenect2
make
make install
sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.d/
```

Índice de Figuras

Índice de Tablas

Bibliografía

[1] *History of QR code*, [consulta: 18 agosto 2021]. Disponible en https://www.qrcode.com/en/history.

[2] *Opencv 2.4.8 documentation*, [consulta: 5 abril 2021]. Disponible en https://docs.opencv.org/2.4.8/.

[3] *ROS*, [consulta: 29 febrero 2021]. Disponible en https://www.ros.org/.

[4] *TurtleBot*, [consulta: 23 agosto 2021]. Disponible en https://www.turtlebot.com.

[5] *Zoom Óptico y zoom digital: ¿cuál es mejor?*, marzo 2021, [consulta: 28 marzo 2021]. Disponible en https://www.dzoom.org.es/zoom-optico-y-zoom-digital-cual-es-mejor.

[6] Francisco Javier Buenavida Durán, *Programación de robot móvil con manipulador para el sector del comercio en entorno ROS*, Sevilla, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla:, 2017.

[7] IEEE, *IEEEE Standard for Information Technology 802.3af*, 2003.

[8] Elise Lachat, Hélène Macher, Tania Landes, and Pierre Grussenmeyer, *Assessment and calibration of a rgb-d camera (kinect v2 sensor) towards a potential use for close-range 3d modeling*, MDPI (2015).

[9] Adrián Fernández Sola, *Desarrollo software del robot turtlebot en el entorno retail*, Sevilla, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla:, 2017.

[10] Antonio Barrientos y otros, *Fundamentos de robótica*, McGraw-Hill, 2007.